TD1 Asm Cortex-M3

Pascal Acco pour l'équipe des branquignolles

Résumé

Après une petite mise en bouche sur les sections de données et la taille des transferts. Vous traduirez un algorithme écrit en langage C pour le stm32 utilisant le GPIO. Cet algorithme peut être très facilement adapté au cœur conçu en SFO : vous aurez ainsi un exemple de programme en langage d'assemblage dont les instructions (ou leur équivalentes) formeront le jeu d'instruction minimal de votre cœur.

I. Petites questions rapides

Dans le module dummy.s figurent les déclarations et le code suivant :

```
CAKE    equ 10
;-------------------------------------------------------------------
    AREA    DummyVars, DATA, READWRITE
Caracteres  DCB     12, -1, 'A'
Nombre      DCW     -2, 0xACDC,     ((CAKE<<12) + (0xC<<8) + (12<<0x4))
Miam_Un     SPACE   CAKE
Fin_Cake

;-------------------------------------------------------------------
    AREA    DummyCode, CODE, READONLY

main PROC

    MOV     R2, #CAKE
    LDR     R1, =Fin_Cake
    LDR     R0, =DummyVars
____Repete
        LDRH    R7, [R0]
        LDRSH   R8, [R0]
        ADD     R0, #2
    B ____Repete

 ENDP ; fin du main
```

L'éditeur de lien décide que `dummyVars` vaut 0x20000000. Indiquez ce que vallent les registres R0, R7, R8 à chaque passage sur l'instruction `B Repete` . Que vallent R2 et R1 ?

II. Problème : Jeu de réflexe

Le programme suivant utilise le GPIO du STM32F103RB, puce incluant le cortex-M3 et des périphériques, qui est similaire dans son fonctionnement à celui que vous avez conçus en SFO. La seule différence est que le CR utilise 4 bits pour configurer une seule broche du port car la configuration est plus fine qu'un simple input ou output. Le jeu de réflexe utilise deux boutons poussoirs (PORT.0 et PORT.3) et deux diodes (PORT.1 et PORT2). Les joueurs attendent à l'affut que les diodes se mettent à clignoter et appuient le plus rapidement possible sur leur bouton poussoir. Dès ce moment seule la diode du joueur le plus rapide reste allumée pendant un certain temps pour indiquer le gagnant. Les diodes s'éteignent ensuite et les joueur retournent aux aguets...

```c
#define IN 0x8 // input with pulldown
#define OUT 0x2 // output 2MHz with push/pull

unsigned int * Cr = (unsigned int *) 0x40010C00; // CRL of GPIOB (STM32f10x)
unsigned int * Idr = (unsigned int *) 0x40010C08; // ODR
unsigned int * Odr = (unsigned int *) 0x40010C0C; // IDR

unsigned char Boutons_Lus;

void Wait(int);

int main(void)
{   unsigned char Pas_Encore ;
    unsigned int mask;

    // configure |PORT.3       |PORT.2        |PORT.1        |PORT.0
    //     as    |In           |Out           |Out           |In
    //___________|_____________|______________|______________|___________
    //  for      |Player1      |Player1       |Player 2      |Player 2
    //           |Button       |Led           |Led           |Button
    mask =        (IN<<(3*4)) +(OUT<<(2*4)) +(OUT<<(1*4)) +(IN<<(0*4));
    *Cr = mask;

    while(1)
    {   Pas_Encore = 'R'; // "Run" car les joueurs n'ont pas encore appuyé sur une touche

        // Fait clignoter les LEDs en attendants que les players appuient
        while (Pas_Encore)  // rappel : 0=> false ; tout le reste (et donc 'R')  => true
        {   // Teste si au moins un bouton est appuyé : on teste donc si IDR.0 (bouton player 1)
            //    ou IDR.3 (bouton player 2 ) est à 1, grâce au masque binaire 2_00001001 = 0x9
            Boutons_Lus = *Idr & 0x9 ;  // mets tous les bits à 0 sauf les bits .0 et .3

            // Boutons_Lus vaut 0 (!False) si aucun bouton appuyé  et true sinon
            if (Boutons_Lus) Pas_Encore =0;

            // Change état des diodes avec un XOR bit à bit du masque 2_00000110
            *Odr ^= 0x6    ;
        }//while pas appuyé

        //etteinds les diodes
        *Odr &=  ~(6); //~  est l'inversion de chaque bits

        // affiche le vainqueur ou ex-aequo
        if (Boutons_Lus & (1<<0)) *Odr |= (1<<1);
        if (Boutons_Lus & (1<<3)) *Odr |= (1<<2);

        // attends qu'ils aient vu le resultat
        Wait(1000000) ;

        //etteinds les diodes
        *Odr &=  ~(6); //~  est l'inversion de chaque bits

        // les joueurs attendent le signal pour appuyer
        Wait(3000000) ;
    }// while 1
} // main

// Boucle de temporisation
void Wait( int N)
{   volatile int cpt ;

    cpt=0;
    for (cpt=0;cpt<N;cpt++);
}
```

Traduisez cet algorithme en assembleur en indiquant l'association entre registres et variables du programme.